A GRID-BASED PATH PLANNING APPROACH FOR A TWO VEHICLE TEAM WITH LOCALIZATION CONSTRAINTS

A Thesis

by

MARK GARBER

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Sivakumar Rathinam
Committee Members,	Swaroop Darbha
	Lewis Ntaimo
Head of Department,	Andreas A. Polycarpou

August 2017

Major Subject: Mechanical Engineering

ABSTRACT

This research proposes a path-finding method for two unmanned vehicles with localization constraints using a modified shortest-path algorithm. A beacon vehicle has GPS or other absolute positioning information, and a target vehicle has only bearing information taken relative to the beacon vehicle or known stationary landmarks. A method for calculating edge costs is described based on factoring the covariance associated with an Extended Kalman filter. By gridding the region and discretizing the position error, the path-planning problem for two vehicles can be formulated in state space and solved using a dynamic programming algorithm. To improve the computation time of the algorithm, a heuristic is also introduced. In simulation, paths found from the dynamic programming method and heuristic consistently outperform a greedy algorithm and find paths that favor localizable regions and result in relatively low amounts of error.

NOMENCLATURE

Acronyms

EIF	Extended Information filter
EKF	Extended Kalman filter
GPS	Global Position System
IMU	Inertial Measurement Unit
$SLAM$	Simultaneous Localization and Mapping
UV	Unmanned Vehicle

Model and Simulation

B	Control Jacobian
$error$	Error level
F	System Jacobian
H	Measurement Jacobian
h	Bearing measurement equation
inc	Error level increment
P	Covariance Matrix
Q	Covariance of control input noise
t_s	Timestep (s)
X_k	State vector at step k
Y	Information Matrix
$\hat{y}_k$	Information vector

ψ	Vehicle heading (rad)
μ_w	Controller error
σ_v	Velocity input standard deviation
σ_w	Turn rate standard deviation
$\sigma_{\psi,0}$	Initial heading standard deviation
λ	Eigenvalue
η_{ij}	Bearing measurement from vehicle i w.r.t. j

Shortest Path Algorithm

Q	Priority queue of current set
R	Stores vertices reserved for future exploration
C	Stores previously explored vertices
P	Path composed of ordered sequence of vertices
v	Vertex

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Professors Sivakumar Rathinam and Swaroop Darbha of the Department of Mechanical Engineering and Professor Lewis Ntaimo of the Department of Industrial and Systems Engineering.

The Matlab simulation model used in the Simulation section was provided by Dr. Rajnikant Sharma of the University of Cincinnati.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

Graduate study was supported by a Research Assistantship from Texas A&M University. This material is based upon work supported by the National Science Foundation under Grant No. 1527748.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

TABLE Page

1. INTRODUCTION

Over the past several years, unmanned vehicles (UVs) have seen increased usage in underwater, ground-based, and aerial applications. With uses ranging from pipeline and power line inspection to border surveillance to search and rescue, UVs are suitable for a wide variety of tasks. One common scenario is for a human operator to be in charge of a team of vehicles. The operator assigns high level missions and objectives, which the vehicles attempt to complete autonomously. There are many subproblems associated with completing these missions. Included in these are localization, path planning, and control. The vehicles must be able to gather information about their location, either through GPS or by sensing their environment. They must be able to use that information with a control system that manages vehicle motion, and they need a path planning method that tells them where in the environment to go, and how to navigate toward that location.

Path planning that considers multiple vehicles navigating an environment in the absence of GPS is one problem to consider. Based on the known information about the environment, a path is selected for the vehicles before they begin travel. The path should avoid states where the position of the vehicles is highly uncertain. Localization is important for UVs, because GPS information is not always available. In indoor environments, it rarely is. Even outdoors, it is not guaranteed. GPS signals are easily jammed, both intentionally and on accident. In one incident, a personal GPS jammer in a truck, intended to prevent an employer tracking the vehicle, interfered with air traffic control systems at Newark airport.

This research will consider a fundamental path planning problem with two vehicles. One vehicle receives GPS information, while the other does not. By utilizing bearing measurements to known landmarks, the vehicles can improve position estimates based on

dead reckoning. Additionally, the vehicle receiving GPS information communicates its location to the vehicle without GPS information to use as an additional landmark. The bearing sensors on the vehicles can only take measurements within a limited range. This paper will develop a grid-based method that can be used to plan paths for each vehicle to manage position uncertainty of the vehicle lacking GPS information [1].

The current literature in this field either considers path planning for a single vehicle in an environment with observable landmarks, or path planning for multiple communicating vehicles without additional sensor information about the environment. While observability analysis has been done which considers both multiple vehicles and available landmarks in the environment [2], path planning while considering localization constraints based on the expected uncertainty in the vehicles' positions has not. This research presents a possible formulation for a path planning problem with two communicating vehicles. The environment contains stationary known landmarks, which the vehicles are able to take bearing measurements of to estimate their relative position. An algorithm to solve the problem is presented, as well as an associated heuristic. A vehicle model with an Extended Information filter to estimate position is described, as well as a method to estimate the error associated with the position. Finally, results based on the algorithm and heuristic are discussed.

2. LITERATURE REVIEW

Over the past decade, there has been a substantial interest in control and path planning for unmanned vehicles (UVs). UVs are useful for a wide variety of applications ranging from power line inspection [3] to air quality measurements [4] to border surveillance [5]. One specific area of research related to UVs has been path planning when faced by localization constraints. Often, UVs will have absolute position information from GPS measurements. In some cases, however, GPS information may not be available due to either operating in a GPS denied environment, or a lack of GPS capability on the UV. In lieu of GPS, other localization techniques may be employed. The simplest technique is dead-reckoning, which extrapolates an estimate of the vehicles current position relative to a previous known or estimated position based on velocity estimates from an internal accelerometer or IMU (Inertial Measurement Unit). While simple to implement, error is propagated quickly in a dead-reckoning estimate. Calibration errors and noise in controllers and IMU information mean that initially small errors grow rapidly, rendering dead-reckoning alone useless for many applications that require localization estimation. To overcome this, dead reckoning position estimates are often fused with other techniques in an attempt to create more robust localization techniques. One popular technique, Simultaneous Location and Mapping (SLAM), generates a map of the environment and its current position as it travels. SLAM often operates using computer vision or range measurements, and often requires additional onboard processing equipment. However, its versatility, especially in a feature-rich environment, has made a popular technique. Other localization techniques attempt to combine the dead-reckoning estimate with other measurements using filtering or probabilistic techniques. Measurements may be taken relative to landmarks of known location, or multiple UVs may be working together and sharing information to perform

cooperative localization in order to minimize error propagation. Several works deal with the observability of vehicles when using such techniques.

In a 2D reference frame, three independent measurements typically are necessary for complete observability. However, in some configurations, two beacons is sufficient to localize a moving vehicle [6]. Several works ([7, 8, 9, 2]) provide similar analysis of the local observability when a system is not fully observable. Even when not sufficient for full observability, bearing measurements can provide a significant improvement in state estimation [7].

When performing localization, there is always an uncertainty associated with the position and orientation estimate, or pose, of the vehicle. With GPS measurements, this uncertainty may be relatively constant, but when relative localization techniques are used, the uncertainty depends on the previous uncertainty used to generate the current position estimate, and the uncertainty associated with the measurements used to estimate the current position. In addition to knowing the position estimate, it is also important to know the uncertainty associated with the position estimate. Combining the estimate and uncertainty produces a region of varying size where the UV is likely to be, centered at the maximum likelyhood location, which is useful for path planning and collision avoidance.

Mourikis and Roumeliotis [10] develop techniques to predict upper bounds on positioning uncertainty for networks of mobile robots performing cooperative localization. If a single vehicle in the network has access to GPS or other absolute positioning information, the uncertainty in the position uncertainty of all the vehicles in the network converges with time, and one can find an upper bound on this uncertainty. However, if there is no absolute information available and only relative position measurements are made, the error bounds will continue to increase over time. Relative position measurements consist of both a bearing and range measurement, as well as an absolute orientation measurement. Hence, for this system, a single relative position measurement is sufficient for complete observability.

When path planning with restricted GPS information, it is often necessary to consider localization constraints and reduce position uncertainty. In these cases, the path planning problems include objectives or constraints to maximize observability, or to bound the uncertainty. Several papers address the problem of single vehicle path planning with localization constraints [11, 12]. In some cases, Simultaneous Localization and Mapping (SLAM) techniques are used in conjunction with path planning techniques to determine an optimal path [13, 14]. Prentice and Roy develop a Belief Roadmap for a region with known landmarks with nodes determined by Probabilistic Random Sampling. In order to reduce computational time, the covariance of the Extended Kalman Filter is factored and a transfer function to calculate covariance in a single step is developed [12].

Multi-vehicle path planning has also been addressed [15, 16], where some vehicles are denied absolute positioning information. These methods typically attempt to maintain complete observability of the vehicles and deal with the paths of the vehicles relative to each other rather than to the final location, which can result in longer paths and may not be feasible when the vehicles have separated origin or goal waypoints. Additionally, these methods typically only consider measurements relative to each other, and do not attempt to take advantage of measurements to external landmarks that may be available to improve localization.

3. PROBLEM STATEMENT

This fundamental path planning problem considers a top-down view of two vehicles traversing a region. A target vehicle and beacon vehicle must each travel from specified starting locations to specified goal locations. The target vehicle does not have GPS, and so it must rely on the landmarks or other nearby vehicles to localize its position. The beacon vehicle is equipped with GPS and can assist the target vehicle in localization while the vehicles travel. The path of both the target and the beacon vehicles are not initially specified. The objective of the problem is to find optimal paths for both the vehicles such that the maximum uncertainty in estimating the position of the target vehicle along the is minimized, and the total path length is also considered. In order to simplify the problem, the motion of the vehicles is restricted, and graph-based approach is used. A grid overlays a map of the environment containing known landmarks. Each vertex of the grid is used as a possible waypoint for both the target vehicle and the beacon vehicle. Vehicles can travel up, down, left, or right to neighboring vertices on the grid. To further limit the complexity of the problem, only one vehicle may move at a time. While one vehicle is traveling, the other vehicle will loiter at its current location. Once a vehicle reaches another vertex, it can travel again, or it can loiter while the other vehicle travels (Fig. 3.1). Only one vehicle at a time is allowed to occupy any given waypoint. Because the problem deals with localization uncertainty, it is also necessary to account for this in the graph-based approach. Depending on the position of the target vehicle relative to the beacon vehicle and fixed landmarks, the position uncertainty of the target vehicle will change. Position uncertainty is discretized into ranges referred to as error levels. When the final uncertainty of the target vehicle position is calculated after traveling an edge, the error is rounded up to the nearest error level, based on a preselected error increment size inc as shown in Fig.

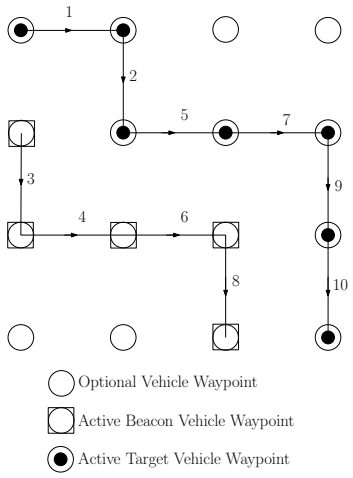

Figure 3.1: Ordered Target and Beacon Vehicle Paths.©2017 IEEE [1]

3.2. Each vertex of the graph will consist of the target vehicle location, the beacon vehicle location, and the error level of the target vehicle. Fig. 3.3 shows possible changes in the error level as the target and beacon vehicles traverse the first section of the path in Fig. 3.1.

The beacon vehicle is assumed to have absolute positioning information, such as from GPS, and negligible position error. The target vehicle does not have any absolute position information. At each timestep it utilizes an Extended Kalman Filter (EKF) to first predict its current state with accelerometer measurements and previous state estimates, then updates that prediction with bearing readings from landmarks or the beacon vehicle, if in range.

Error levels are determined based on the Extended Kalman filter. The maximum eigenvalue of the covariance matrix corresponding to position error generated by the EKF is used to determine the error level. The objective is to plan a path for the target and beacon vehicles such that they each arrive at their respective goal destinations, and the maximum intermediate error of the target vehicle is as low as possible. This is accomplished using a

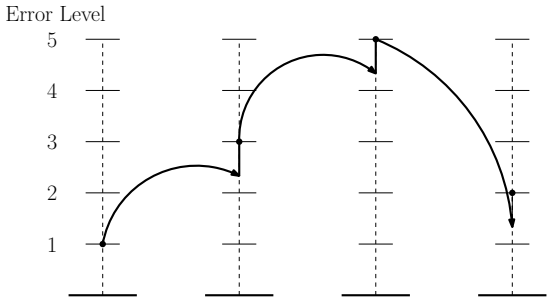

Figure 3.2: Position uncertainty is rounded up to the next error level.

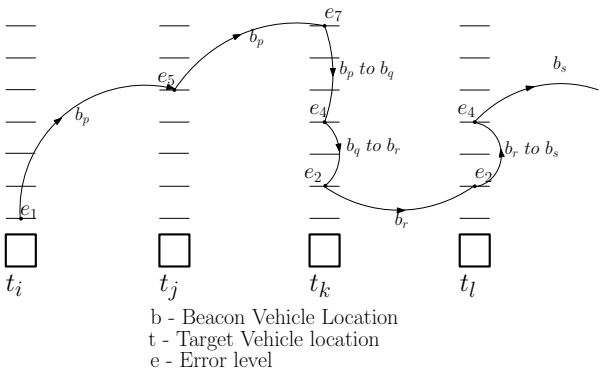

Figure 3.3: Error levels as target and beacon vehicles travel.©2017 IEEE [1]

shortest path algorithm that starts by searching vertices at the lowest error level and adds higher error level vertices until a feasible path is found. The method in this paper does not directly consider observability, however observability does effect the localization error, which is considered. Vehicles will likely travel through regions with fewer than two bearing measurements available, and depending on landmark distribution may never be fully observable [1].

4. MODEL DESCRIPTION

A Matlab model is used to simulate vehicle travel. The model was utilizes a variant of an Extended Kalman Filter to predict the vehicle states, then update them based on external sensor information [17]. Both the target and beacon vehicle positions are determined in this manner, however, the beacon vehicle receives absolute position information at each timestep, while the target vehicle receives only bearing information. The true state of the two vehicles is $X_k = [x_t\ y_t\ \psi_t\ x_b\ y_b\ \psi_b]^T$, where x and y are the position, and ψ is the heading of the target vehicle t and the beacon vehicle b. There is also $\hat{X}_k$, the state estimated by the filter. The state at timestep $k+1$ can be written as

$$X_{k+1} = X_k + \begin{bmatrix} V_{t,k} * cos(\psi_k) \\ V_{t,k} * cos(\psi_k) \\ \psi_{t,k+1} \\ V_{b,k} * cos(\psi_k) \\ V_{b,k} * cos(\psi_k) \\ \psi_{b,k+1} \end{bmatrix} \tag{4.1}$$

With no turn radius constraint, the controller always attempts to set the heading towards the next waypoint. In order to estimate current state, an Information Filter (IF) is used. The IF is a variant on the Extended Kalman filter, and employs prediction and update steps to first estimate the position based on the the previous position and IMU information about velocity and orientation, then updates that estimate with any available bearing information. If bearing information is available, the filter will combine the information from the position and update steps in a way that minimizes the covariance. Using an Extended Information

filter instead of a standard Extended Kalman filter makes calculating multiple updates for a single timestep faster, but is mathematically identical to the Extended Kalman filter. The information vector $\hat{y}_k$ and information matrix Y_k are related to the state estimate $\hat{X}_k$ and covariance matrix P_k of the Extended Kalman filter as follows:

$$Y_k = P_k^{-1} \tag{4.2}$$

$$\hat{y}_k = Y_k \hat{X}_k \tag{4.3}$$

The Information filter consists of two steps. The first step predicts the state based on the previous state and IMU information.

$$Y_{k+1|k} = (F_k Y_{k|k}^{-1} F_k^T + B_k Q_k B_k^T)^{-1} \tag{4.4}$$

$$\hat{y}_{k+1|k} = Y_{k+1|k} \hat{X}_{k+1|k} \tag{4.5}$$

$$\hat{X}_{k+1|k} = X_{k|k} + t_s f(\hat{X}_{k|k}, u_k) \tag{4.6}$$

F_k is the system Jacobian and is derived from 4.1 to be

$$F_k = \begin{bmatrix} F_1 & 0 \\ 0 & F_2 \end{bmatrix} \tag{4.7}$$

where

$$F_i = \begin{bmatrix} 1 & 0 & -V_i t_s sin(\psi) \\ 0 & 1 & V_i t_s cos(\psi) \\ 0 & 0 & 1 \end{bmatrix} | i = 1, 2 \tag{4.8}$$

$$B_k = \begin{bmatrix} t_s cos(\psi_{t,}k) & 0 \\ t_s sin(\psi_{t,k}) & 0 \\ 0 & t_s \\ t_s cos(\psi_{b,k}) & 0 \\ t_s sin(\psi_{b,k}) & 0 \\ 0 & t_s \end{bmatrix} \tag{4.9}$$

$$Q = \begin{bmatrix} \sigma_v^2 & 0 \\ 0 & \sigma_w^2 \end{bmatrix} \tag{4.10}$$

If bearing measurements are available, the update step is performed. Bearing measurements μ_{ij} are taken relative to the current position and heading of vehicle i and the position of landmark or vehicle j. In the simulation, they are measured as:

$$\eta_{ij} = \tan^{-1}\left(\frac{y_j - y_i}{x_j - x_i}\right) - \psi_i \tag{4.11}$$

The bearing estimate $h_{ij}(x_{k+1|k})$ is similar to the measurement η_{ij}. For bearing estimates between the two vehicles i and j,

$$h_{ij} = \tan^{-1}\left(\frac{\hat{y}_j - \hat{y}_i}{\hat{x}_j - \hat{x}_i}\right) - \hat{\psi}_i \tag{4.12}$$

For bearing estimates between a single vehicle i and a landmark j of a known position,

$$h_{ij} = \tan^{-1}\left(\frac{y_j - \hat{y}_i}{x_j - \hat{x}_i}\right) - \hat{\psi}_i \tag{4.13}$$

The update step is:

$$Y_{k+1|k+1} = Y_{k+1|k} + \sum H_{ij|k}^T \sigma_{ij}^{-1} H_{ij,k} \tag{4.14}$$

$$\hat{y}_{k+1|k+1} = \hat{y}_{k+1|k} + \sum H_{ij,k} R^{-1} \times (\mu + H_{ij,k} \hat{\hat{X}}_k) \tag{4.15}$$

where $\mu_{ij} = \eta_{ij} - h_{ij}(x_{k+1|k})$ is the difference between the measured bearing η_{ij} and estimated $h_{ij}(x_{k+1|k})$ bearing of landmark or vehicle j relative to vehicle i.

H_{ij} is the measurement Jacobian, and is defined as:

$$H_{ij|k} = \frac{\partial h_{ij}}{\partial X}|_{X=X_k} \tag{4.16}$$

For bearing estimates from the vehicle i to vehicle j (the target and beacon vehicles, in either order)

$$H_{ij} = [H_{ij,1}\ H_{ij,2}]' \tag{4.17}$$

$$H_{ij,1} = \left[\frac{(\hat{y}_j - \hat{y}_i)}{(\hat{x}_j - \hat{x}_i)^2 + (\hat{y}_j - \hat{y}_i)^2}, \frac{-(\hat{x}_j - \hat{x}_i)}{(\hat{x}_j - \hat{x}_i)^2 + (\hat{y}_j - \hat{y}_i)^2}, -1 \right] \tag{4.18}$$

$$H_{ij,2} = \left[\frac{-(\hat{y}_j - \hat{y}_i)}{(\hat{x}_j - \hat{x}_i)^2 + (\hat{y}_j - \hat{y}_i)^2}, \frac{(\hat{x}_j - \hat{x}_i)}{(\hat{x}_j - \hat{x}_i)^2 + (\hat{y}_j - \hat{y}_i)^2}, 0 \right] \tag{4.19}$$

For bearing estimates from vehicle i to landmark j,

$$H_{ij} = \left[\frac{(y_j - \hat{y}_i)}{(x_j - \hat{x}_i)^2 + (y_j - \hat{y}_i)^2}, \frac{-(x_j - \hat{x}_i)}{(x_j - \hat{x}_i)^2 + (y_j - \hat{y}_i)^2}, -1 \right] \tag{4.20}$$

The sequence of prediction and update steps produces an estimate of the position of each vehicle at each timestep, and the corresponding covariance matrices provide a measure of the uncertainty associated with that estimate.

GPS information is provided to the beacon vehicle as an additional measurement, where

$$H_{GPS} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{4.21}$$

and the measurement are the coordinates of the beacon vehicle. This model was used in Matlab [17], and the error estimation is based off of the covariance of the Extended Information filter.

5. ERROR LEVEL ESTIMATION

5.0.1 Overview and Model-Based Estimation

As discussed in the previous chapter, a discrete time system is used for a vehicle model, and an information filter is used to improve positioning accuracy by using bearing estimates. The simplest approach to determining error levels is by using the same model that is used for simulation. The simulation is run over each edge, with position uncertainty of $inc \times level$ used to initialize the covariance matrix, where inc is the error increment and $level$ is the error level associated with the starting vertex, up to a maximum error level n. Initial heading uncertainty $\sigma_{\psi,0}$ is held constant across all error levels. The final error of the edge is determined by $\sqrt{\lambda_{max}}$, where λ_{max} is the largest eigenvalue of the covariance matrix corresponding to position uncertainty. Each edge is simulated multiple times and the median covariance used. One limitation of using position covariance as the measure of uncertainty is that σ_ϕ is held constant. Error growth in the prediction stage of filtering is primarily dependent on orientation and velocity error. However, in simulation, orientation uncertainty follows the same patters as position uncertainty. When position uncertainty is kept reasonably low, orientation uncertainty also remains low. In simulation, assuming a reasonable constant initial uncertainty produces reasonable results, but does mean that the final error level is not a bound.

If the target and beacon vehicles occupy the same location on the grid, or a vehicle will travel outside the grid after traveling along an edge, the edge is not created. Up to eight edges are created from each vertex, with the target or beacon vehicle traveling to an adjacent waypointwhile the remaining vehicle loiters.

If the error level matrix is generated ahead of time in Matlab via simulation, a value must be assigned for every edge, even those that it may not be possible to reach. Calculat-

ing the error level matrix is the computationally expensive part of the problem. An X by Y grid with n error levels will have approximately $4(XY)^2 * n$ edges to simulate. Many of these edges are duplicates that need only be calculated once, but the cost of finding the error level matrix increases rapidly with the number of edges. For the 15x8 grids used in simulations, calculating the error levels takes approximately 90 minutes per grid depending on landmark density. Error levels were calculated in Matlab running parallel loops on a quad core Intel i5 CPU. In addition to the runtime, the resulting array also requires a sizeable amount of memory.

5.0.2 Factoring the Covariance

To improve the performance of the algorithm and reduce the resource requirements for the error estimation, it is desirable to develop a faster function that can be called inside the algorithm each time an error level is needed. Because error levels are computed only as they are needed in the algorithm presented, it is not necessary to generate and store a matrix containing the error levels for every single possible edge. Even for the integer programming formulation, which requires all error levels ahead of time, the computation below is faster than Matlab simulation. The first step is to simplify the model. By making some assumptions about the beacon vehicle, a more computationally efficient algorithm can be developed. Recall that while it is subject some controller error, the uncertainty in the positioning of the beacon vehicle is always very low due to GPS information. If the beacon vehicle position error is assumed negligible in simulation, the beacon vehicle can be treated as another landmark. If the target vehicle is traveling and the beacon vehicle is loitering, the beacon vehicle can be treated as a single stationary landmark. If the beacon vehicle is traveling in direction ϕ with velocity V_j while the target vehicle loiters, the

beacon vehicle can be treated as a landmark whose position at timestep T is given as

$$X_j = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + T * \begin{bmatrix} V * t_s * \cos(\psi) \\ V * t_s * \sin(\psi) \end{bmatrix} \tag{5.1}$$

where $[x_0\ y_0]^T$ is the starting point of the beacon vehicle along the edge. If $V_j = 0$, the equation reduces to the stationary case $[x_0, y_0]'$.

Using the simulation model to calculate error levels has several drawbacks. The results it produces vary if an edge is simulated multiple times, it can be difficult to incorporate individual edge cost calculation directly into the algorithm, meaning that the entire matrix of error levels may need to be pre-generated, and the simulation may be slower than other methods. For the model used in this research, the inverse of 6×6 matrices must be taken at every step, which is an expensive operation. Performance can be improved by implementing part of the results found in [12] and factoring the covariance updates. This removes the need for matrix inverses at each timestep and producing consistent results independent of random noise in the system.

In order to implement the factored covariance, the target vehicle is assumed to be at its maximum likelihood position (x_i, y_i). When is moving, it is assumed to be along the straight edge, and when loitering, it is assumed to hold a constant position. When combined with the assumption about beacon vehicle position described previously, this means that the position of both vehicles is assumed (although there is still an uncertainty associated with the target vehicle position). A summary of the derivation and results from [12] are included below.

From [18],

Lemma 1.

$$(A + BC^{-1})^{-1} = (ACC^{-1} + BC^{-1})^{-1} = C(AC + B)^{-1} \tag{5.2}$$

Theorem 2. *The covariance matrix P of the target vehicle can be factored as $P = UV^{-1}$, where U_{k+1} and V_{k+1} are found from the Extended Kalman Filter process as linear functions of U_k and V_k*

Proof. Proof by induction

Base Case. The theorem is trivially true, as

$$P_0 = U_0 V_0 = P_0 I^{-1} \tag{5.3}$$

Induction Step Given

$$P_k = U_k V_k^{-1} \tag{5.4}$$

Recall equation 4.2 and 4.4

$$Y_k = P_k^{-1}$$
$$\bar{Y}_k = (F_k Y_k^{-1} F_k^T + B_k Q_k B_k^T)^{-1}$$

then

$$\bar{P}_{k+1} = F_{k+1} P_k F_{k+1}^T + B_{k+1} Q_{k+1} B_{k+1}^T \tag{5.5}$$

$$\text{Let } R_k = B_k Q_k B_k^T \tag{5.6}$$

$$\bar{P}_{k+1} = F_{k+1} U_k V_k^{-1} F_{k+1}^T + R_{k+1} \tag{5.7}$$

$$\bar{P}_{k+1} = (F_{k+1} U_k)(F_{k+1}^{-T} V_k)^{-1} + R_{k+1} \tag{5.8}$$

From 5.2

$$\bar{P}_{k+1} = \left(\left(F_{k+1}^{-T} V_k \right) \left(F_{k+1} U_k + R_{k+1} \left(F_{k+1}^{-T} V_k \right) \right)^{-1} \right)^{-1} \tag{5.9}$$

$$\bar{P}_{k+1} = \left(\bar{D}_{k+1} \bar{E}_{k+1}^{-1} \right)^{-1} \tag{5.10}$$

$$\Rightarrow \bar{P}_{k+1} = \bar{E}_{k+1} \bar{D}_{k+1}^{-1} \tag{5.11}$$

Where $\bar{D}_{k+1} = F_{k+1}^{-T} V_k$ and $\bar{E}_{k+1} = F_{k+1} U_k + R_{k+1} \left(F_{k+1}^{-T} V_k \right)$.

This factored form of $\bar{P}_{k+1}$ is preserved while maintaining the update process. Similarly, recalling equation 4.14

$$\bar{Y}_{k+1} = \bar{Y}_k + \sum H_{ij,k}^T \sigma_{ij}^{-1} H_{ij,k}$$

Then

$$\bar{P}_{k+1} = \left(\bar{P}_k^{-1} + \sum H_{ij,k}^T \sigma_{ij}^{-1} H_{ij,k} \right)^{-1} \tag{5.12}$$

Letting $M_k = \sum H_k^T \sigma_t^{-1} H_k$ and substituting in equation 5.11

$$P_{k+1} = \left(\bar{D}_{k+1} \bar{E}_{k+1}^{-1} + M_{k+1} \right)^{-1} \tag{5.13}$$

From 5.2

$$\bar{P}_{k+1} = \bar{E}_{k+1} \left(\bar{D}_{k+1} + M_t \bar{E}_{k+1} \right)^{-1} \tag{5.14}$$

$$\Rightarrow \bar{P}_{k+1} = U_{k+1} V_{k+1}^{-1} \tag{5.15}$$

where $U_k = \bar{E}_k$ and $V_k = \bar{D}_{k+M_k \bar{E}_k}$. Collecting terms,

$$U_{k+1} = \bar{E}_k = \bar{F}_{k+1} U_k + R_{k+1} (F_{k+1}^{-T} V_k) \tag{5.16}$$

and

$$V_{k+1} = \bar{D}_{k+1} + M_{k+1}\bar{E}_{k+1} \tag{5.17}$$

$$= F_{k+1}^{-T}V_k + M_{k+1}(F_{k+1}U_k + R_{k+1} + R_{k+1}(F_{k+1}^{-T}V_k)) \tag{5.18}$$

Collecting terms again, the expression can be rewritten as,

$$\begin{bmatrix} U \\ V \end{bmatrix}_{k+1} = \underbrace{\begin{bmatrix} 0 & I \\ I & M \end{bmatrix}_{k+1}}_{A_{1|k+1}} \underbrace{\begin{bmatrix} 0 & F^{-T} \\ F & RF^{-T} \end{bmatrix}_{k+1}}_{A_{2|k+1}} \begin{bmatrix} U \\ V \end{bmatrix}_k \tag{5.19}$$

□

The factorization method can be initialized using equation 5.3. Note that only M in A_1 changes as an edge is traversed, and A_2 remains constant along an edge. Furthermore, in order to calculate A_1, it is not necessary to take a matrix inverse. The final covariance can then be calculated as

$$P_k = U_k V_K^{-1} \tag{5.20}$$

where

$$\begin{bmatrix} U & V \end{bmatrix}_k = A_{1|k}A_2 * A_{1|k-1}A_2 * ... * A_{1|1}A_2 * \begin{bmatrix} P_0 \\ I \end{bmatrix} \tag{5.21}$$

Calculating error using this method is significantly faster than finding it directly from the simulation, and provides consistent results.

6. INTEGER PROGRAM FORMULATION

One possible approach is to formulate and solve the path planning problem as an integer program. The integer program considers edges connecting adjacent vertices and finds a set of active edges that connect the origin state to the goal state. The first edge of the path should start with the target and beacon vehicles at their respective origin positions, and the last edge should end with the target and beacon vehicles at their respective goal positions. The intermediate edges should form a continuous path (Fig. 3.3).

There are two types of edges. A beacon edge refers to an edge where the beacon vehicle travels from i to j, the target vehicle loiters a waypoint u, and the error level of the target vehicle changes from x to y. A beacon edge is expressed as β_{ijuxy}. A target edge refers to an edge where the target vehicle travels from u to v, the beacon vehicle loiters at waypoint i, and the error level of the target vehicle changes from x to y. A target edge is expressed as τ_{iuvxy}. Target edges and beacon edges are binary, taking a value of one if the edge is part of the optimal solution, and zero otherwise. The uncertainty level of the target vehicle at its origin waypoint is initially assumed to be one, although this can be changed in the problem formulation. The origin set S_{o1} consists of all target and target edges entering an origin vertex. The set S_{g1} consists of all target and target edges entering a goal vertex. Similarly, the sets S_{o2} and S_{g2} consists of the edges exiting an origin vertex and a goal vertex, respectively.

An integer programming formulation of the path planning problem is shown below.

$Minimize\ T$

$s.t.$

$$-T + y * \tau_{iuvxy} \leq 0 \qquad \forall t \tag{6.1}$$

$$-T + y * \beta_{ijuxy} \leq 0 \qquad \forall b \tag{6.2}$$

$$\sum_{u,x} \tau_{juvxy} + \sum_{i,x} \beta_{ijvxy} - \sum_{u,x} \tau_{jvuyx} - \sum_{i,x} \beta_{jivyx} = 0 \qquad \tau, \beta \notin S_{o2} \ \ and\ \tau, \beta \notin S_{g1} \tag{6.3}$$

$$\sum_{\tau \in S_{o2}} \tau + \sum_{\beta \in S_{o2}} \beta = 1 \tag{6.4}$$

$$\sum_{\tau \in S_{g1}} \tau + \sum_{\beta \in S_{g1}} \beta = 1 \tag{6.5}$$

$$\sum_{\tau \in S_{o2}} \tau + \sum_{\beta \in S_{o1}} \beta = 0 \tag{6.6}$$

$$\sum_{\tau \in S_{g2}} \tau + \sum_{\beta \in S_{g2}} \beta = 0 \tag{6.7}$$

$\tau,\ \beta\ are\ binary$

This objective minimizes the maximum error level of the path. Once T is found, the program can be run again with the shortest path objective and an additional constraint on the vertex error level. Constraints 6.1 and 6.2 define T, the variable in the objective used to minimize the maximum error level. Constraint 6.3 states that for all intermediate vertices, the number of edges entering a vertex must be equal to the number exiting. A vertex is entered when either a beacon or target vehicle travels to a new waypoint resulting in the state corresponding to the vertex, and exited when either vehicle travels again resulting in a new state. Constraints 6.4 and 6.5 state that either a target edge or beacon edge must

exit the origin vertex, and either a target edge or beacon edge must enter the final vertex. Finally, constraints 6.6 and 6.7 prevent loops back to the origin and goal vertices that would satisfy 6.3 instead of forcing a complete path by preventing edges from entering an origin vertex or exiting a goal vertex.

This formulation finds the lowest possible maximum error level. Modifying the objective to $Minimize\ T + \alpha(\sum \tau + \sum \beta)$ for $\alpha \ll 1$ finds the shortest path whose maximum error level does not exceed the maximum error level T associated with the original objective value.

This program was implemented and tested using the Julia programming language and Gurobi. It produced paths with objective values identical to the paths produced by the shortest path algorithm. However, the solution time for finding an optimal solution using this integer programming approach increased rapidly as the number of waypoints on the grid was increased. In the next section, we propose a faster, more efficient algorithm to directly solve the path planning problem with a single goal for each vehicle.

7. ALGORITHM*

Algorithm 1 Shortest Path Algorithm

```
1:  procedure SHORTESTPATH
2:      create vertex PriorityQueue Q
3:      create vertex sets C[], R[]
4:      for level = 1 : numlevels do
5:          for r in R[level] do
6:              Add r to Q with priority dist[r] + length(r, goal)
7:          add v(t_o, b_o, level) to Q with priority 0
8:          while Q is not empty do
9:              remove lowest priority vertex u from Q
10:             if u is a goal vertex then
11:                 return prev, dist, u
12:             for all vertices v adjacent to u do
13:                 if v.e ≤ numlevels then
14:                     cost = dist[u] + length(u, v)
15:                     est = cost + length(v, goal)
16:                     if v is not in C[v.error] then
17:                         if v.e ≤ level then
18:                             add v to Q with priority est
19:                         else
20:                             add v to R[v.error]
21:                         dist[v] = cost
22:                         prev[v] = u
23:                         add v to C[v.error]
```

Algorithm 1 Continued

```
24:                 else
25:                     if cost < dist[v] then
26:                         dist[v] = cost
27:                         prev[v] = u
28:                         if v is in Q then
29:                             update priority of v to est
30:                         else if v.e ≤ level then
31:                             add v to Q with priority est
```

The path planning problem can also be formulated as a dynamic program. A modified shortest path algorithm described here solves the dynamic problem by exploring adjacent vertices in an efficient way to produce an equivalent solution to the integer program. * The algorithm is a modified version of A^*, but the entire accessible region will be searched at each error level before searching higher error levels. Limiting the searched region is discussed later in this section. This algorithm runs much faster than the integer programming formulation.

The positions of the target and beacon vehicles are placed as vertices on a directed graph. The vertices's states are composed of $[t, b, error]$, where t is the target vehicle position, b is the beacon vehicle position, and $error$ is the error level of the target vehicle. For two connected vertices, either t or b will change, but not both. $error$ is based on the error estimation results. An origin vertex is a vertex $[t_o, b_o, error]$ where t_o and b_o are the origin locations of the target and beacon vehicles, and $error$ is any error level. Similarly, a goal vertex is any vertex $[t_{goal}, b_{goal}, error]$ where t_{goal} and b_{goal} are the goal locations of the target and beacon vehicles, and $error$ is any error level. Typically the optimal path will contain an origin vertex at error level one. If the initial position is uncertain, the algorithm

*Reprinted with permission from "A grid-based path planning approach for a team of two vehicles with localization constraints" by M. Garber, S. Rathinam and R. Sharma, 2017. 2017 International Conference on Unmanned Aircraft Systems (ICUAS), Miami, FL, USA, 2017, pp. 516-523, Copyright 2017 by IEEE.

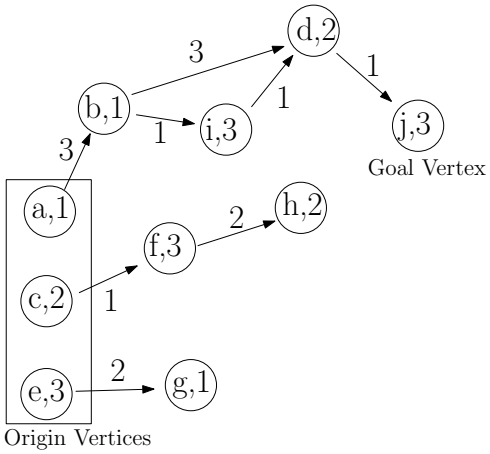

Figure 7.1: Example of a graph solved using the shortest path algorithm. Nodes are visited in order from a to k. Error levels are shown in each vertex. All edge costs are one. ©2017 IEEE [1]

can be modified to start with a higher error level.

7.0.1 Implementation Details

Fig. 7.2 shows the process of the algorithm. The algorithm is initialized by setting the active error level to one and adding the origin vertex with the active error level to the priority queue Q. If the initial error level is higher, the active error level can be adjusted appropriately. After initialization, the vertex u with the lowest estimated cost is removed from Q and explored (initially the origin vertex). For this problem, cost is measured by length of the path, with all edges having length one. The estimated cost is the length of the path to the current vertex, plus the shortest path following the grid edges from the current vertex to the goal locations. $length(i, j)$ is the sum of the shortest possible path for the target and beacon vehicles traveling along grid edges from vertex i to vertex j.

Vertices v_i that can be reached directly from u are examined. Any v_i with no current cost, or with a current cost greater than the sum of current cost of u and the cost from u to v_i is updated to reflect the lower cost and corresponding path. If the error level of an

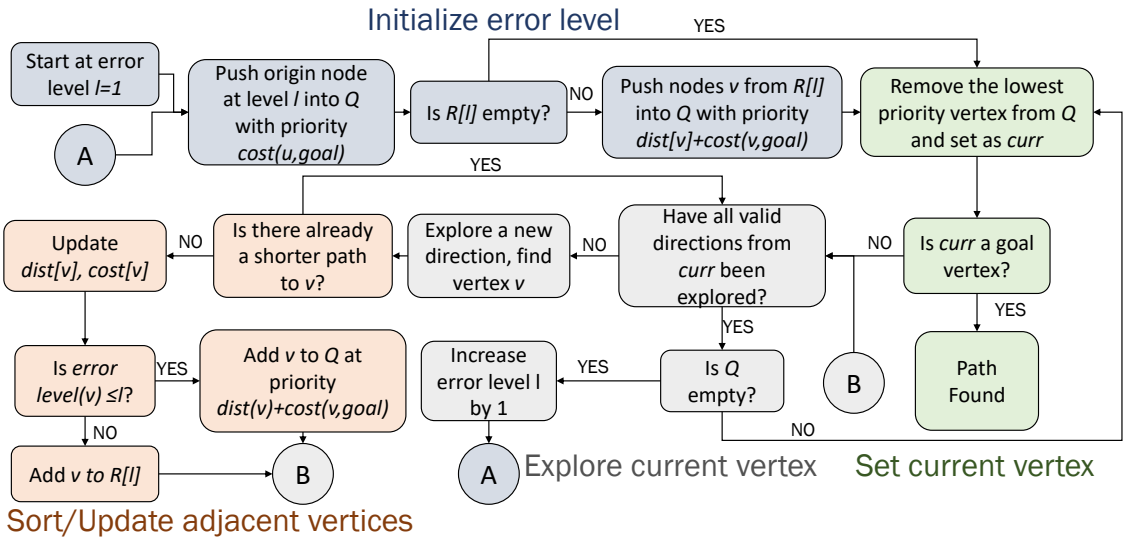

Figure 7.2: Flow chart of the algorithm process.

updated v_i is less than or equal to the active error level, v_i is added to Q if not already present, and its priority in Q is updated to be equal to its new estimated cost. Otherwise, it is saved for later exploration in $R[v_i.error]$. Once all the vertices directly reachable from u are explored, a new u is removed from Q. This process is repeated until Q is empty, or a path to a goal vertex is found. If no path is found, the active error level is increased by one. Previously discovered vertices in $R[active\ error\ level]$ with error levels equal to the new active error level are added to Q, and the graph is explored at the current error level. $C[]$ is used to track which vertices have been explored.

Fig. 7.1 and Table 7.1 shows an example path explored using the algorithm. Vertex f is explored twice. It is initially discovered and explored at error level two. A shorter path to f is found at error level three, so f is explored again. The final path traverses $g - h - f - i - j$ to reach the goal vertex.

Table 7.1: Order of vertex exploration in Fig.7.1.©2017 IEEE [1]

Error Level	Vertices Explored
1:	a, b
2:	c, d, e, f
3:	g, h, f, i, j

The algorithm described finds a path for which the error level of all vertices in the path is equal to or less than the current error level. The path found will be the shortest path from an origin vertex to a goal vertex using only the vertices at or below the current error level.

With minor changes to the algorithm, the objectives and constraints associated with the problem can be changed. A path length constraint can be added by not considering vertices that exceed an estimated final cost. An initial error level greater than one may be specified, and if multiple paths below that error level exist, the shortest will be found. The stopping criterion can be changed so that only the target vehicle has a goal destination. Finally, the search could be continued to a set number of error levels above the lowest error level with a feasible solution in order to search for shorter paths.

7.0.2 Algorithm Summary

The standard A^* algorithm attempts to find the shortest path to a goal vertex. If it fails to find a solution, it will explore all accessible nodes. Because the heuristic used in this problem to estimate remaining cost can never exceed the true remaining cost, A^* is guaranteed to find the shortest path, if it exists. However, because this problem also considers minimizing the error level, a standard shortest path algorithm such as A^* cannot be used directly.

The objective for this problem is to first minimize the maximum error level of the target vehicle, then minimize the total path length for that maximum error level. The path edges for target and beacon vehicle travel can be weighted differently, but will always be positive, that is $length(v_x, v_y) > 0$ for all connected vertices v_x and v_y. For paths simulated in this article, all edges are weighted equally. Paths begin at any origin vertex v_o and consist of an ordered sequence of connected vertices. Every explored vertex has a previous vertex associated with it that is part of its shortest path. By tracing back the previous vertices from the goal vertex to the origin vertex, the optimal path can be found once the algorithm is run.

So long as each vertex is connected to the previous vertex in its shortest path, the algorithm is guaranteed to find an optimal solution. However, this is dependent on searching vertices in order of lowest cost. There are two differences between the standard A^* algorithm with a priority queue and the one presented here. First, vertices are added to the queue as they are discovered, so only vertices accessible from an origin vertex are explored. As the vertices in any path must be accessible an origin vertex (all paths start at the source and all vertices in any path are therefore necessarily accessible from an origin vertex), this does not affect results. Second, additional vertices are added if the vertices that are accessible with the current error level have all been explored and no path to an origin vertex has been found. This addition of new vertices may allow access to previously inaccessible vertices, as well as creating shorter paths to previously explored vertices.

Newly available unexplored vertices, either just added or previously inaccessible, will be explored as normal. If the new vertices result in a new optimal path for a previously explored vertex v_i, the cost and optimal path for v_i are updated as normal. However, any vertices directly accessible from v_i must also be checked again. If the v_i is part of the previously optimal path for some adjacent vertex v_j, the path of v_j is updated implicitly with v_i. However, the cost of v_j must be updated as well. If v_i was not previously part

of v_j's optimal path, but it becomes part of the optimal path due to the reduced cost of v_i, then both the path and cost of v_j must be updated. Since v_j has also been updated, it must also be added again to the queue and re-explored. This can result in chains of updated vertices that must be explored again when the active error level is increased. By re-adding vertices to the priority queue whenever they are updated and exploring them along with other vertices in order of cost, optimality of the algorithm is maintained [1].

7.0.3 Problem Setup and Tuning Parameters

There are several factors that contribute to the success of the algorithm. A maximum path length, l_{max}, can be added to limit the length of the path relative to its maximum possible length. If the total estimated cost of a vertex v_i being explored exceeds l_{max}, that is, the sum of cost of the current best path to v_i and the shortest path cost from v_i to the goal is greater than l_{max}, then v_i is not added to the priority queue Q. v_i may be revisited later if a shorter path to v_i is found at a higher error level. This serves the dual purposes of adding a length constraint and reducing time wasted exploring directions that lead further from the goal, especially at error levels which must be fully explored but do not contain a complete path from origin to goal. Typically if the target and beacon vehicles start and end close to each other, a 1.2-1.5 times the shortest possible combined path with not have an effect on the final path. In a few cases, this may not be true, but these cases should be apparent from simple visual inspection.

A second parameter to be considered is the error increment size inc. A large inc decreases the fidelity of the error estimation, meaning that two paths that result in a significantly different final error are rounded to the same error level and treated equivalently in the algorithm. This can lead to inferior solutions that would be significantly improved through use of a smaller error level. On the other hand, setting the error level too low can lead to constant backtracking, and number of error levels explored before a final solution

is reached may increase substantially, although the number of vertices explored per error level will be lower. Backtracking can often be discouraged by setting a low maximum path length, or forbidding returning to the previous target and beacon location. A large number of error levels becomes a major limitation if it is necessary to generate error levels ahead of time, as explored vertices are not known in advance and so error levels must be calculated for all edges. If the error levels are calculated within the algorithm, the number of error levels may not be an issue, although the larger number of explored vertices may still result in an increased runtime. Edge length or grid size is also an important factor to consider. While larger edge lengths will reduce the number of vertices that must be explored in the graph, it also further restricts travel of the target and beacon vehicles. If the edge length is greater than the bearing sensor range, the target vehicle will never be close enough to the landmark vehicle to receive bearing information. The edge length should be less $Range/\sqrt{(2)}$, where $Range$ is the range of the bearing sensor. Smaller edge lengths will improve results, but also increase the number of vertices and the time to run the algorithm.

8. HEURISTICS

While the shortest path algorithm presented in the previous section prodduces good paths in a variety of situations, it can become expensive quite quickly. The upper limit of the solve time is proportional to grid size squared, or linear proportional to the total number of target and beacon location combinations. This means that, while the shortest path algorithm is able to handle smaller problems, it can become impractical for path planning over larger regions. One approach to handling this is to develop heuristics that, while they may not find the best path, still find a satisfactory path. Towards that end, a heuristic is presented and discussed.

8.0.1 Fixed Target Path Heuristic

The heuristic attempts to improve solve time by breaking the problem into two steps. The first step disregards the beacon vehicle entirely, and only considers finding a path for the target vehicle. In the second step, the modified shortest path algorithm is run, but the target vehicle is only allowed to visit waypoints that are on the path found in the first step. This type of algorithm typically results in paths with a comparable maximum error to the shortest path algorithm, but slightly longer paths.

There are two possible approaches to finding the initial target vehicle path. The first approach, levels-based, is to solve the problem using the shortest path algorithm, but without the beacon vehicle. Error levels are still used, and a vertex consists of the target vehicle position and error level. The algorithm essentially attempts to minimize the the longest section of the path without localization information from landmarks.

The second option is to solve the target vehicle path problem as a single layer shortest path problem using A^*. In this case, the cost of an edge is the final error after traveling along that edge. Error is determined using the same methods described in the previous

sections, but is not rounded up to the closest error level. All edges are assumed to start at the same error level, regardless of what the final error of other edges connecting to the edge.

The first approach has the advantage of attempting to limit the maximum error over the entire path. This is a more conservative approach than the single layer shortest path problem, which may choose a more direct path and depending how far from the direct path they are, ignore all landmarks entirely. The single layer approach solves slightly faster since there are not multiple error levels to be considered, but it also does not place as much emphasis on the shortest path. In general, both approaches perform similarly and produce paths similar to the direct algorithm. Typically, if a heuristic is necessary the single layer path should be used first because of the faster solve time. If the result is not satisfactory, the levels based approach may be used.

Once a target vehicle path is determined, the original modified shortest path algorithm is run. However, only vertices in which the target vehicle remains on the path determined in the first step of the algorithm are explored. The beacon vehicle is still free to roam across the grid. Because the number of possible target vehicle waypoints is reduced, the total number of vertices is also reduced from $(\#\ of\ waypoints)^2 * (max\ error\ level)$ to $(\#\ of\ waypoints) * (target\ path\ length) * (max\ error\ levels)$. Since the beacon vehicle will typically remain close to the target vehicle when possible (otherwise traveling towards the goal), the number of these possible vertices that are actually explored drops further.

This heuristic typically produces results with similar maximum error and slightly longer paths than the original algorithm. In some cases the path may actually be shorter if the error level is higher. However, the heuristic will never be able to find a path with a lower maximum error level than the shortest path algorithm.

9. SIMULATION*

Simulation was performed in Matlab with an EKF. 10 meter grid edges were used, with increments of 0.1 meters per error level ($inc = 0.1$). The sensor range on the target vehicle was 25 meters. The maximum error level was minimized, and the shortest path was found that did not exceed the maximum error level.*

9.0.1 Comparison to Greedy Algorithm

For comparison, a greedy algorithm was used to generate an alternate path. The states for the greedy algorithm consist of the target and beacon vehicle locations $[t, b]$. The greedy algorithm (1) selects the directions of travel along the grid that decrease the total distance to the desired state, up to two for each vehicle, (2) selects the vehicle and direction of travel from the previous step that has the lowest error level and penalties associated with it, and (3), updates the current state and repeats steps (1)-(3) until reaching the destination.

In order to remain feasible, the greedy algorithm keeps a constant initial error level at all vertices and uses the error levels of candidate vertices as costs. A penalty is added to the error level proportional to the distance between the target and beacon vehicles to increase the likelihood that they will remain within range. Because target vehicle travel typically results in higher error levels than beacon vehicle travel, another penalty is added to error levels for beacon vehicle travel. If a penalty is not used, the beacon vehicle will travel to its goal before the target vehicle begins to travel.

The greedy algorithm works best if the origins and goals of the target and beacon vehicles are close to each other. If the origins or the goals are separate the vehicles may not even be able to get within range of each other. Furthermore, the vehicles can not move

*Reprinted with permission from "A grid-based path planning approach for a team of two vehicles with localization constraints" by M. Garber, S. Rathinam and R. Sharma, 2017. 2017 International Conference on Unmanned Aircraft Systems (ICUAS), Miami, FL, USA, 2017, pp. 516-523, Copyright 2017 by IEEE.

towards each other or regions with more bearing information if it means moving away from their goal waypoints. If the target vehicle moves toward another landmark, it may be impossible for the landmark and target vehicles to move back towards each other.

When finding a greedy path, if the two vehicles start separated, or are drawn apart because the target vehicle follows stationary landmarks to lower error, they are unable to find each other again and the advantage of cooperative localization is lost, and the target vehicle only receives bearing information from stationary landmarks. Because of this, the number and location of landmarks more heavily influences localization accuracy when using the greedy algorithm. The optimized algorithm can attempt to ensure the target vehicle is always close to the beacon vehicle or a stationary landmark, while the greedy algorithm often must rely on the beacon vehicle or landmarks being within nearby. By following a longer path, the localization uncertainty over the path can be decrease. Conversely, the length of the path generated by the greedy algorithm will always be the shortest path possible (while following the grid), but the localization error may be much higher [1].

An example of paths generated by the greedy algorithm and optimized algorithm as well and their position errors is shown in Figs. 9.1 and 9.2. In the greedy path, the target and beacon vehicles alternate travel over most of the path, until position 24, when the target vehicle travels continuously to its goal destination. The beacon vehicle then completes its path. Vehicles often get separated when localization information from another landmark is available,and are unable to reconnect. Vehicles take the shortest path possible along the grid. Over the path from the modified shortest path algorithm, the target vehicle takes a longer path in order to take advantage of bearing information from nearby landmarks. While the overall path is longer, and the error is not always lower than that from the greedy algorithm, the path found here does not have the large spike in error present in the greedy algorithm path.

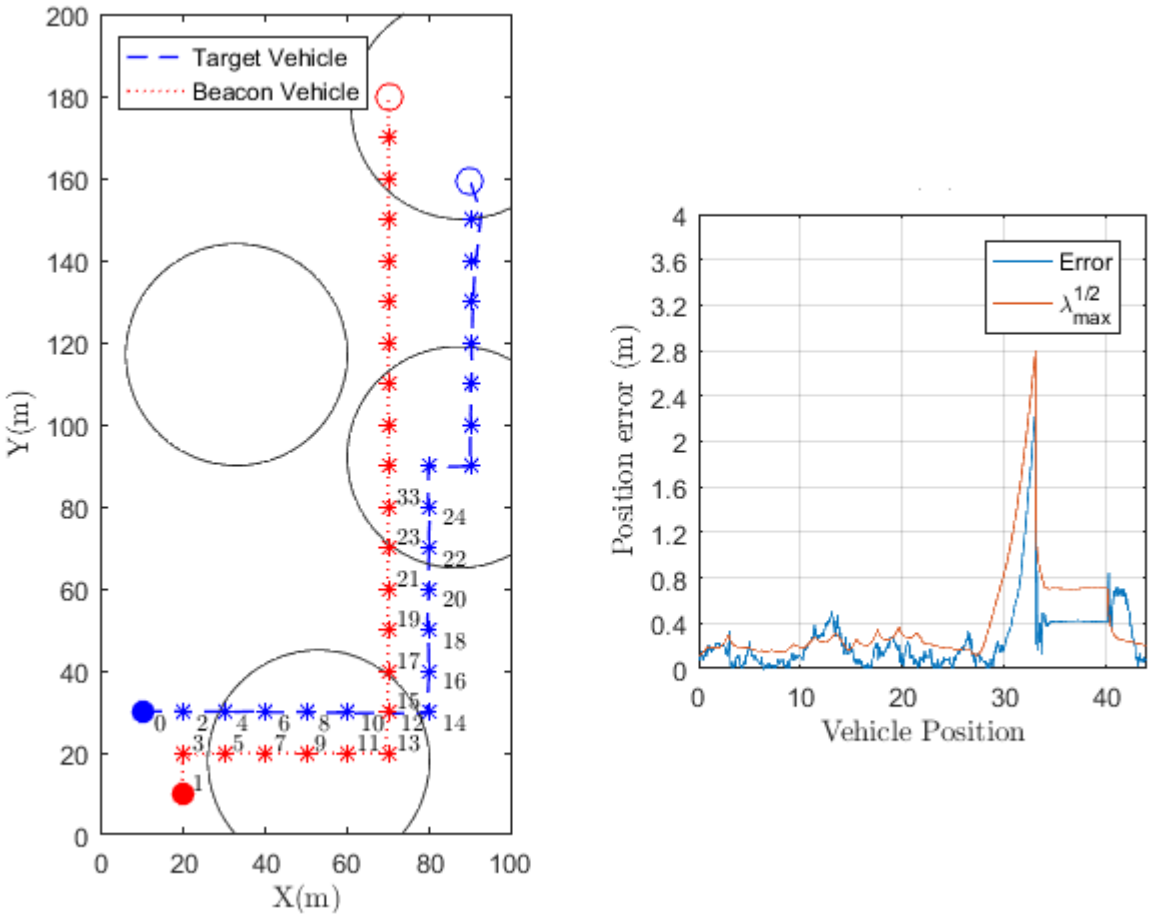

Figure 9.1: Top: Path from greedy path algorithm across a region with scattered landmarks. Numbers indicate vehicle travel beginning at the labeled position. The target vehicle travels two edge lengths from position 0, then loiters while the beacon vehicle travels one edge length from position 2, and so forth. Large circles show the region in landmark range. Bottom: Position error from a single simulation and the position uncertainty from the covariance matrix are shown with vehicle position corresponding to top figure.©2017 IEEE [1]

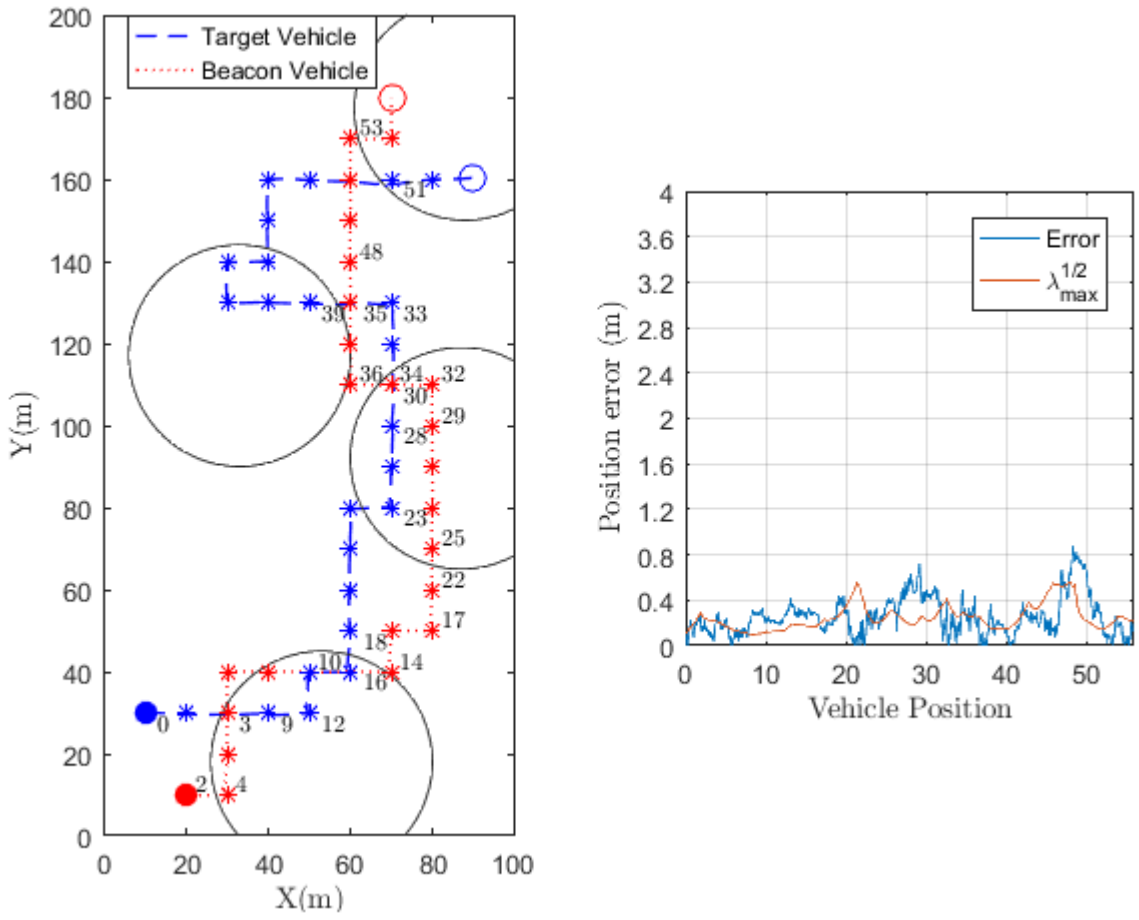

Figure 9.2: Path and error generated by modified shortest path algorithm. Error is similar for most of the path to the greedy algorithm, but does not jump like the greedy algorithm.©2017 IEEE [1]

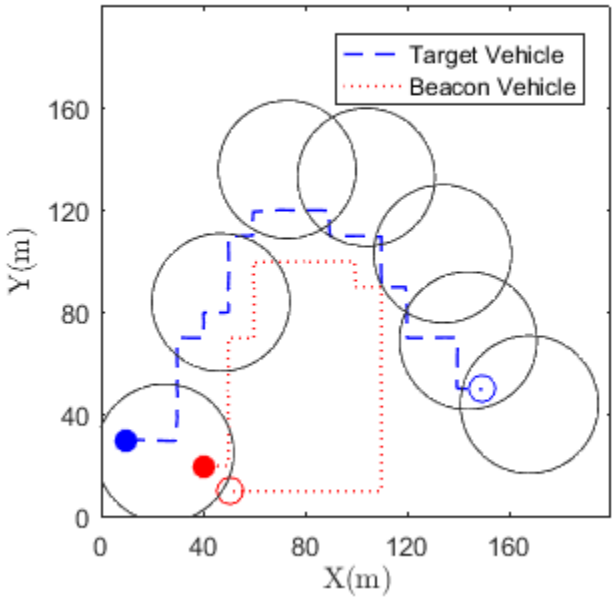

Figure 9.3: Beacon vehicle assisting the target vehicle, then returning near its original location

9.0.2 Beacon Assist Vehicle

Figure 9.3 shows a scenario where the target vehicle has a goal destination across the map, while the beacon vehicle is available to assist the target vehicle, but has its goal close to the origin waypoints of the vehicles. The beacon can also use the same waypoint as its origin in destination, but in the figure, the beacon origin and goal waypoints are separated to more clearly show the path of the beacon vehicle. The beacon vehicle travels far enough to assist the target vehicle with localization as it travels to its destination, and then returns as the target vehicle continues to its destination. Additionally, the target vehicle takes a longer path in order to remain within range of the stationary landmarks.

9.0.3 Fixed Target Path Heuristic

The paths produced by the fixed target path heuristic are typically similar to, but not identical to, those produced by the original algorithm. Figs. 9.5, 9.6 show paths generated by the levels based fixed target path heuristic and single layer fixed target path heuristic. In this example, the path in figure 9.4) from the original algorithm is quite similar to the path from the heuristic with the single layer shortest target path. The path from the levels target path uses the bottom landmark to reduce the length of the longest section of the target vehicle path.

9.0.4 Run Time

Run time is a limiting factor for the shortest path algorithm presented in this paper. As shown in Fig. 9.7, solve time varies significantly with the location of landmarks, but in worst cases increases proportionally to the square of the number of grid points, or linearly with the total number of vertices (composed of a target location, beacon location, and error level). When the error levels of edges are precalculated and provided to the algorithm, the runtime is reduced to less than five percent of the values shown in Fig. 6. The variance in solve time is typically related to the maximum error level of the path. Generally, when a feasible path exists at a low error level, fewer vertices are explored and so the algorithm finds a solution more quickly. Often when the algorithm runs, the first few error levels will have relatively few vertices to be explored, typically between a few hundred and a few thousand, depending on the size of the map and location of the landmarks. At a certain error level, the target and beacon vehicles can travel together over a large region of the grid, and the number of vertices to explore at that and subsequent error levels increases by several orders of magnitude. If a solution is found before that error level, the solve time is relatively low. If a solution is found at that error level, the solve time is higher, but because of the estimated cost heuristic from A^*, it is not necessary to explore the entire

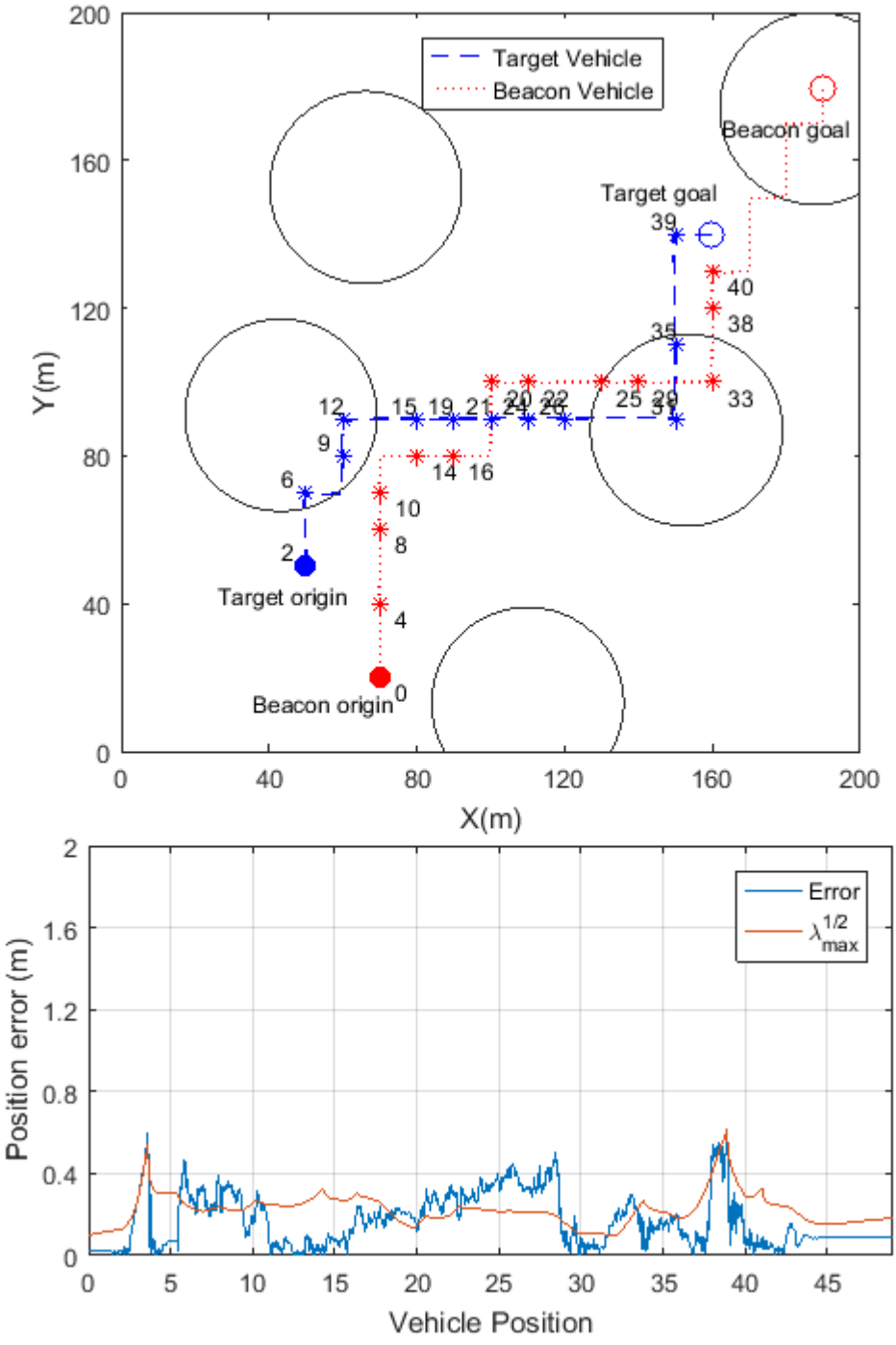

Figure 9.4: Path generated by the modified shortest path algorithm and corresponding simulated error and covariance over a single run.

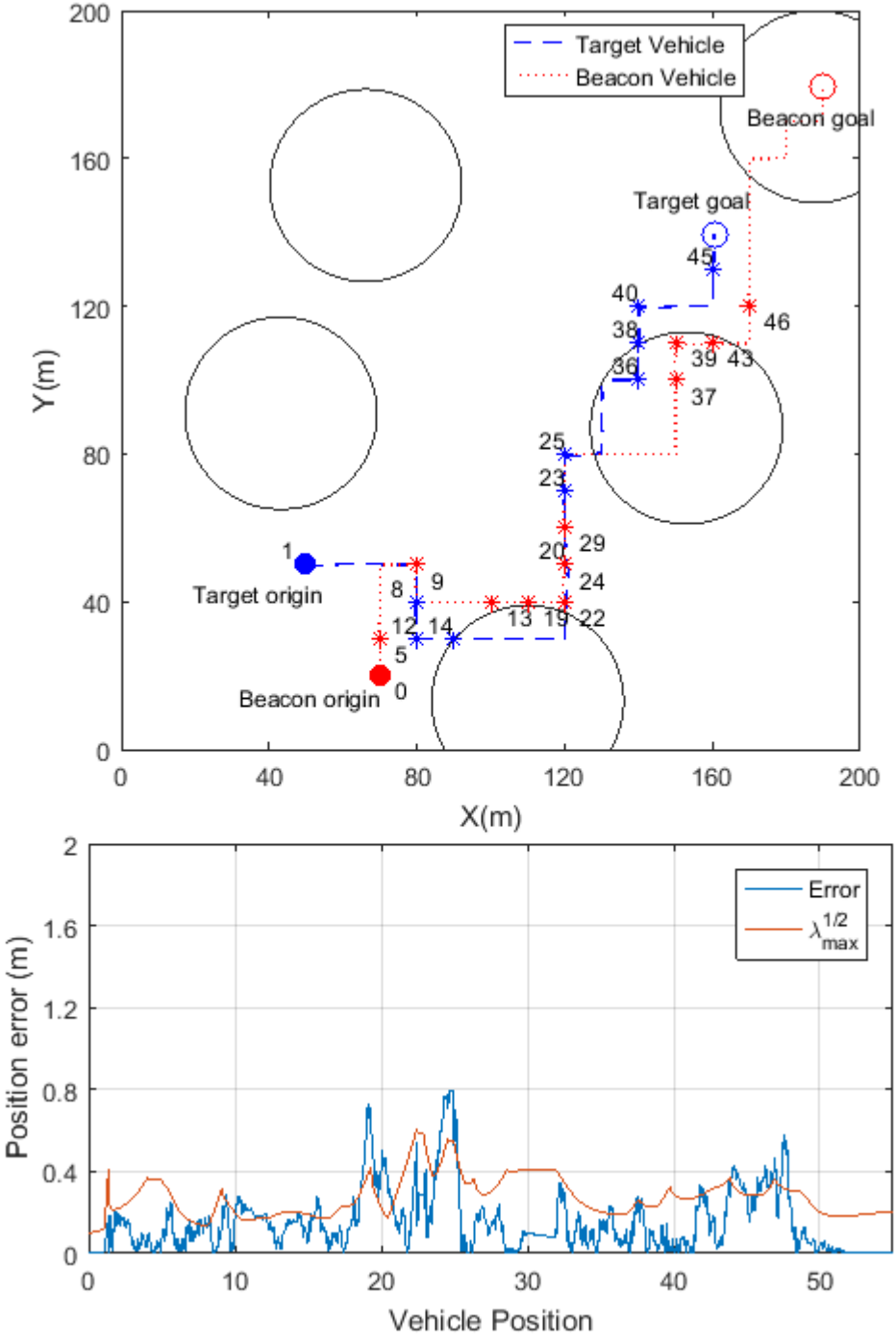

Figure 9.5: Path generated by the levels based fixed target path heuristic and corresponding simulated error and covariance over a single run.

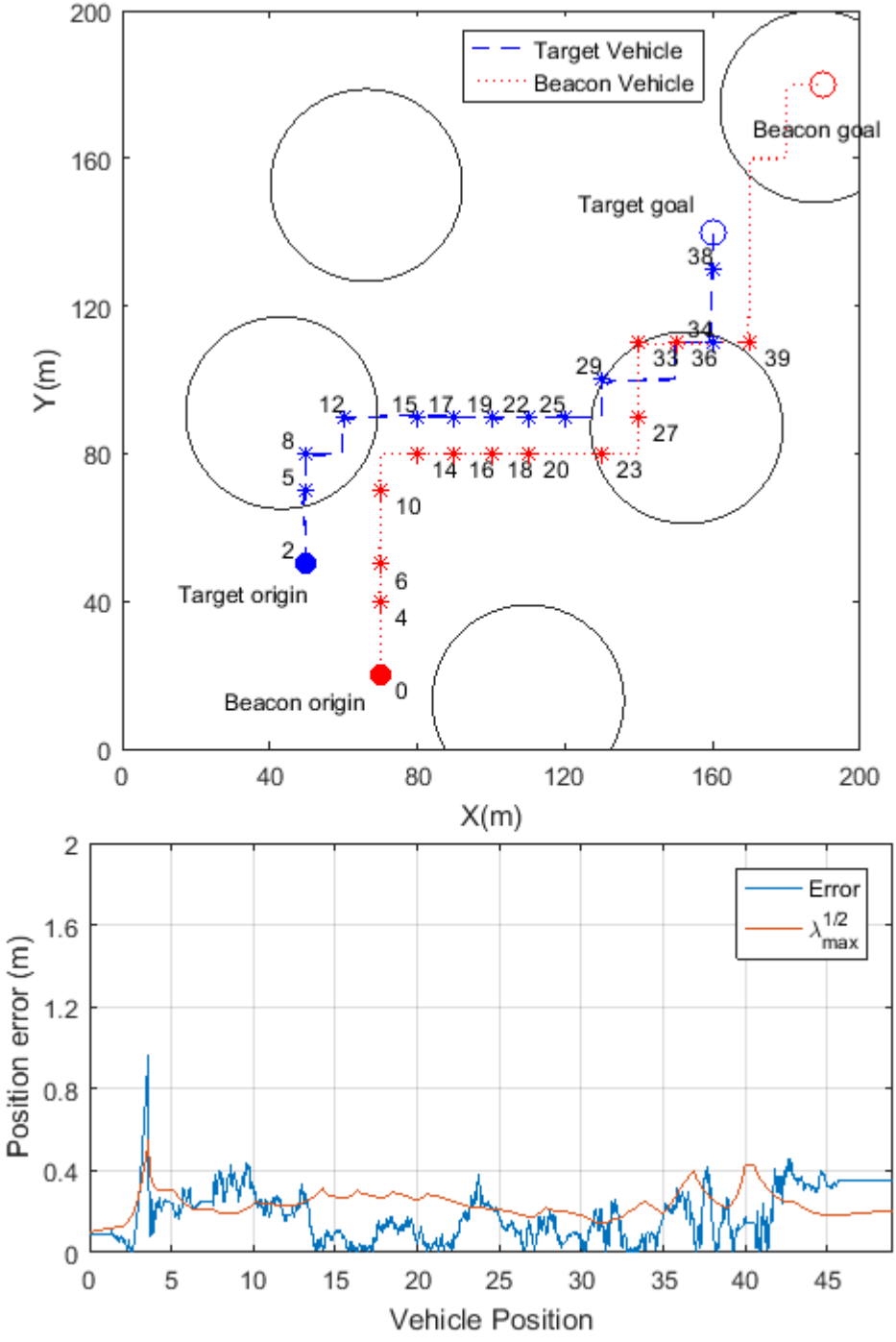

Figure 9.6: Path generated by the single layer fixed target path heuristic and corresponding simulated error and covariance over a single run.

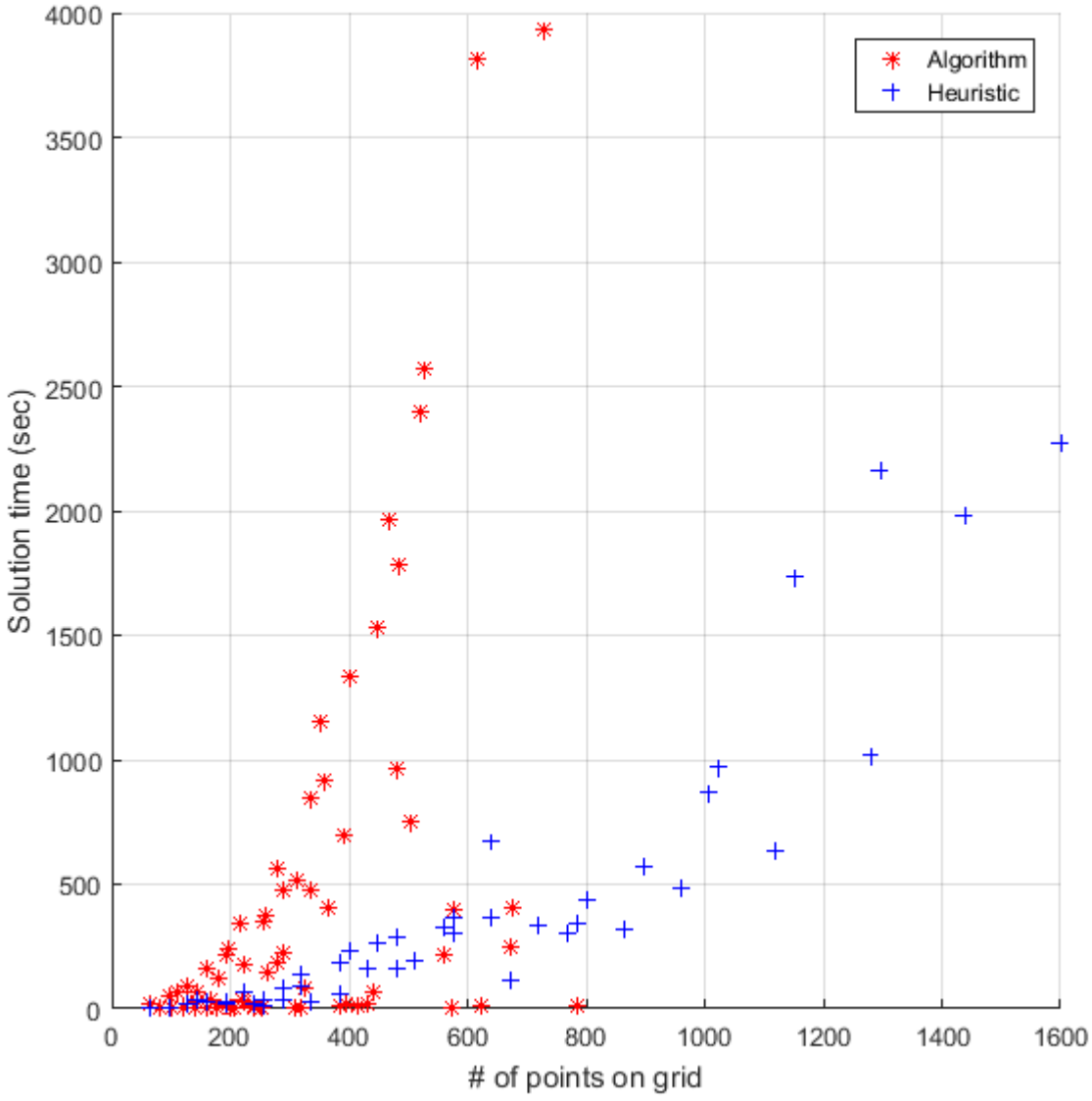

Figure 9.7: Solve time using the dynamic algorithm and fixed single level target path heuristic. Maximum path length was limited to 1.5x the minimum possible path length on the grid for the dynamic algorithm. The greater of 2 or (# of grid points)/50 landmarks were placed randomly on the grid, with a sensor range of 25m.

error level. If the solution exists at an error level after then size of each error level has increased significantly, then the algorithm takes much longer to run.

Using the single level target path heuristic, the solve time can be significantly reduced. Fig. 9.7 shows solve time for the target path heuristic, with the initial target vehicle path found using error as edge costs. Note the difference in scales between the two figures. Around 600 waypoints, the worst case solutions take approximately an hour to solve, while the largest heuristic solve time is 11 minutes, with most grids taking closer to 6 minutes. On the other hand, there were some configurations of landmarks for which the solve time using the original algorithm was also around 6 minutes, or even significantly less. This is by no means guaranteed, however, and the maximum and average solve times using the heuristic are much lower than the shortest path algorithm. The solve time also varies by landmark density. The solve times for Figs. 9.4, 9.5, 9.6, which considered a 20×20 grid with 400 points were 45 seconds using the levels based fixed target path heuristic, 30 seconds using the single layer fixed target path heuristic, and 916 seconds using the modified shortest path algorithm. In these cases, the heuristics found solutions much more quickly than the shortest path algorithm.

10. CONCLUSIONS

This paper proposes a path finding algorithm for two vehicles with localization constraints, as well as a heuristic based on the algorithm. The problem formulation is based on gridding the region, and discretizing the position uncertainty into discrete levels. A method of estimating error based on factored covariance is described. Simulation validates that the methods proposed outperforms a greedy algorithm and finds paths that favor localizable regions and result in relatively low amounts of error.

10.0.1 Limitations

The largest limiting factor is currently the computation costs. The size of the problem grows quickly with the size of the region. As a result of this and the edge cost calculation, the size of the region that can be reasonably considered is limited. Additionally, while the method is well suited to many different landmark densities, it is not always the best choice. By forcing paths along grid edges, directions that may be otherwise preferable are not available, and initial and goal positions are limited to points on the grid. When landmarks are dense, methods that attempt to maximize observability may serve better. If landmarks are too sparse, it may be better to find the shortest length path using landmarks as nodes, and sending the target and beacon vehicle side by side, or traveling in a pattern to maximize observability similar to [15]. Vehicles are also limited to those which have a negligible or near-negligible turning radius

10.0.2 Future Work

The current method is limited by error level calculation costs. Developing or adapting a method to more quickly estimate error levels would allow for solving problems over larger regions with more possible states. Additional, faster-performing heuristics could

also allow for path finding over larger regions. In addition to allowing for larger regions, faster solve times would also allow for expansion of the problem. Additional vehicles can easily be added to the states, or the states can be altered to account for both vehicles traveling simultaneously. Doing so would increase the number of states, but the proposed algorithm could still be applied with minor modifications. Diagonal edges could also be considered, allowing the vehicles more flexibility in their paths. While this research has established one possible framework for approaching the two-vehicle path planning problem described, there is room for significant future improvement and expansion on this work.

REFERENCES

[1] M. Garber, S. Rathinam, and R. Sharma, "A grid-based path planning approach for a team of two vehicles with localization constraints," in *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 516–523, June 2017.

[2] R. Sharma, R. W. Beard, C. N. Taylor, and S. Quebe, "Graph-based observability analysis of bearing-only cooperative localization," *IEEE Transactions on Robotics*, vol. 28, pp. 522–529, April 2012.

[3] Y. Zhang, X. Yuan, Y. Fang, and S. Chen, "Uav low altitude photogrammetry for power line inspection," *ISPRS International Journal of Geo-Information*, vol. 6, p. 14, Jan 2017.

[4] T. F. Villa, F. Gonzalez, B. Miljievic, Z. D. Ristovski, and L. Morawska, "An overview of small unmanned aerial vehicles for air quality measurements: Present applications and future prospectives.," *Sensors (14248220)*, vol. 16, no. 7, pp. 1 – 29, 2016.

[5] A. R. Girard, A. S. Howell, and J. K. Hedrick, "Border patrol and surveillance missions using multiple unmanned air vehicles," in *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*, vol. 1, pp. 620–625 Vol.1, Dec 2004.

[6] P. Bonnifait and G. Garcia, "Design and experimental validation of an odometric and goniometric localization system for outdoor robot vehicles," *IEEE Transactions on Robotics and Automation*, vol. 14, pp. 541–548, Aug 1998.

[7] A. Martinelli and R. Siegwart, "Observability analysis for mobile robot localization," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*,

pp. 1471–1476, Aug 2005.

[8] I. Shames, A. N. Bishop, and B. D. O. Anderson, "Analysis of noisy bearing-only network localization," *IEEE Transactions on Automatic Control*, vol. 58, pp. 247–252, Jan 2013.

[9] S. Wang, L. Chen, H. Hu, and D. Gu, "Single beacon based localization of auvs using moving horizon estimation," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 885–890, Nov 2013.

[10] A. I. Mourikis and S. I. Roumeliotis, "Analysis of positioning uncertainty in reconfigurable networks of heterogeneous mobile robots," in *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, vol. 1, pp. 572–579 Vol.1, April 2004.

[11] S. D. Bopardikar, B. Englot, and A. Speranzon, "Multiobjective path planning: Localization constraints and collision probability," *IEEE Transactions on Robotics*, vol. 31, pp. 562–577, June 2015.

[12] S. Prentice and N. Roy, "The belief roadmap: Efficient planning in belief space by factoring the covariance," *The International Journal of Robotics Research*, vol. 28, no. 11-12, pp. 1448–1465, 2009.

[13] R. Valencia, J. Andrade-Cetto, and J. M. Porta, "Path planning in belief space with pose slam," in *2011 IEEE International Conference on Robotics and Automation*, pp. 78–83, May 2011.

[14] H. K. Nguyen and M. Wongsaisuwan, "A study on unscented slam with path planning algorithm integration," in *2014 11th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, pp. 1–5, May 2014.

[15] Y. T. Tan, R. Gao, and M. Chitre, "Cooperative path planning for range-only localization using a single moving beacon," *IEEE Journal of Oceanic Engineering*, vol. 39, pp. 371–385, April 2014.

[16] M. Suresh and D. Ghose, "Group coordination and path replan tactics in gps denied environments," in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 31–39, June 2016.

[17] R. Sharma, S. Quebe, R. Beard, and C. Taylor, "Bearing-only cooperative localization: Simulation and experimental results.," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 72, no. 3-4, pp. 429–440, 2013.

[18] D. Vaughan, "A nonrecursive algebraic solution for the discrete riccati equation," *IEEE Transactions on Automatic Control*, vol. 15, pp. 597–599, Oct 1970.